

PERFORMANCE ANALYSIS OF THREE DIMENSIONAL
INTEGRAL EQUATION COMPUTATIONS ON
A MASSIVELY PARALLEL COMPUTER

A Thesis

Presented to

The Graduate College

Hampton University

In Partial Fulfillment


of the Requirements for the Degree

Master of Science


by
Terry G. Logan

August 1994

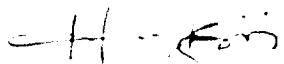
This thesis submitted by Terry G. Logan in partial fulfillment of requirements for the Degree of Master of Science in Applied Mathematics at Hampton University, Virginia, is hereby approved by the committee under whom the work has been done.



Committee Chairman
Dr. Hong Hu



Dr. Arun Verma



Dr. Halima Ali

Dr. Demetrius Venable
Vice President for Research
and Dean of the Graduate College

Date

ACKNOWLEDGEMENTS

First and foremost, I thank God for all the great things he has done for me. Especially for blessing me with a very patient and knowledgeable advisor, Professor Hong Hu. Without his guidance and supervision none of this would have been possible. The suggestions and guidance from my thesis committee members, Professor Halima Ali and Professor Arun Verma, have proven to be invaluable.

This work is supported by the NASA Grant: NAG-1-1170. Walter Silva is the technical monitor. Mr. Silva's moral support and willingness to give me access to resources at NASA-Langley is greatly appreciated.

The computational resources for CM-5 is provided by NASA Numerical Aerodynamics Simulation (NAS) program. The support and suggestions given by Dr. Edward Hook of NAS significantly contributed to this endeavor. His assistance is duly acknowledged and is truly appreciated.

Thank you, Mom, Dad, my brothers, sisters, and the rest of my family for believing in me and constantly encouraging me.

I would also like to thank all of my friends who supported me morally. Especially two friends who are very dear to me and who recently passed on, Deogracias Abella, M.D., December, 1993, and Rochelle Cousins, May, 1994.

I wish to extend special gratitude last, but not least, to my wonderful fiancée, Amber Johnson. Her love, support and encouragement inspire me in every possible way.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	ii
TABLE OF CONTENTS	iii
LIST OF SYMBOLS	v
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF GRAPHS	ix
CHAPTER 1. INTRODUCTION	1
CHAPTER 2. MATHEMATICAL FORMULATION	4
2.1 The Physical Problem	4
2.2 Fundamental Equations of Fluid Flows	4
2.2.1 The Continuity Equation	4
2.2.2 The Momentum Equation	5
2.2.3 The Energy Equation	6
2.3 Full Potential Equation	7
2.4 Boundary Conditions	10
2.5 Integral Equation Solution	11
2.6 Discretization	12
2.7 Numerical Scheme	12
2.7.1 Panel Method for Incompressible Flows	12

2.7.2 Field Panel Method for Compressible Flows	13
2.8 Significance of Computing In a Parallel Environment	15
CHAPTER 3. PARALLEL IMPLEMENTATION	16
3.1 Connection Machine - 5	16
3.2 CM-FORTRAN	17
3.3 Parallel Implementation	18
3.3.1 Parallelization of Incompressible Flow Case	19
3.3.1.1 Attributes of the SPREAD Function	19
3.3.1.2 Attributes of the FORALL Statement	21
3.3.1.3 Attributes of the WHERE Statement	21
3.3.1.4 Attributes of the Parallel Matrix Solver	22
3.3.1.5 Parallelization of Post-Processing Computations	23
3.3.2 Parallelization of Compressible Flow Case	23
CHAPTER 4. PERFORMANCE ANALYSIS	26
4.1 Performance of Incompressible Flow Computations	26
4.2 Performance of Compressible Flow Computations	29
CHAPTER 5. CONCLUDING REMARKS	31
FIGURES	32
TABLES	40
GRAPHS	43
REFERENCES	48

LIST OF SYMBOLS

$[A]$	Influence coefficient matrix
a	Speed of sound
$\{B\}$	Left-hand matrix of the linear system
c	Wing root chord length
C_p	Pressure coefficient
\vec{d}	Distance between receiver (x, y, z) and sender (ξ, η, ζ)
$d\vec{S}$	Elemental surface area
ds	Infinitesimal surface area
e	Internal energy
\vec{e}_d	Unit vector of surface elements
\vec{f}_b	Body forces per unit mass
\vec{f}_v	Viscous forces per unit area
\vec{F}_v	Viscous force
g	Wing surface
G	Full compressibility
κ	Gas specific heat ratio
$M(x, y, z)$	Local Mach number
M_∞	Free-stream Mach number
\vec{n}	Surface unit normal vector
p	Pressure
\dot{q}	Volumetric rate of heat addition per unit mass

q	Source strength
\dot{Q}'_v	Total heat addition to the control volume due to visous effects
\dot{Q}_v	Heat addition to the control volume due to viscous effects
\vec{r}	Position vector
ρ	Density
ρ_∞	Density at infinity
S	Surface of control volume
t	Time
\vec{V}	Velocity vector
$\vec{V}^2/2$	Kinetic energy per unit mass
w	Wake surface
\dot{W}'_v	Total work due to viscous effects on the control surface
\dot{W}_v	Work due to viscous effects on the control surface
Φ	Velocity potential, $\nabla\Phi \equiv \vec{V}$
\forall	Control volume

LIST OF FIGURES

	Page
Figure 1a. Serial version - constructing [A]	32
Figure 1b. CM Parallel version - constructing [A]	33
Figure 2a. Serial version - post-processing calculations	34
Figure 2b. CM Parallel version - post-processing calculations	35
Figure 3a. Serial version - calculating volume integrals	36
Figure 3b. CM Parallel version - calculating volume integrals	37
Figure 4a. Serial version - summing the volume integrals	38
Figure 4b. CM Parallel version - summing the volume integrals	39

LIST OF TABLES

	Page
Table 1. CPU time in second for constructing matrices	40
Table 2. CPU time in seconds for Gaussian elimination	40
Table 3. CPU time in seconds for post-processing	41
Table 4. Total CPU time in seconds for incompressible flow	41
Table 5. CPU time in seconds for evaluating volume integrals	42
Table 6. Total CPU time in seconds for 6 iterations	42

LIST OF GRAPHS

	Page
Graph 1. Effect of number of blocks in Gaussian elimination, $N=24 \times 12$	43
Graph 2. Effect of number of blocks in Gaussian elimination, $N=48 \times 24$	43
Graph 3. CPU time for constructing matrices, $N=48 \times 24$	44
Graph 4. CPU time for evaluating volume integrals, $N=24 \times 12$	45
Graph 5. CPU time for compressible flows with 6 iterations, $N=24 \times 12$	46
Graph 6. CPU time for compressible flows with 6 iterations, $N=48 \times 24$	47

CHAPTER 1

INTRODUCTION

Massively Parallel Processing (MPP) computers offer the prospect of significant performance gains over conventional supercomputers, which are now approaching hard physical speed limits inherent in their technology. However, the need for faster computations continues to grow. As a consequence, massively parallel computers are being developed as a possible solution. The Connection Machine-5 (CM-5) is one such MPP computer, which may provide better performance than conventional supercomputers and may replace the fastest conventional supercomputer in the near future. In fact, the CM-5 computer, with maximum 16k nodes installed, is a 2 TFLOPS computer in theory.

Computational fluid dynamics (CFD) is one of the areas which requires super-fast computational power because the problems typically involve a large number of calculations. The massively parallel computers have the potential to become the main computational tool for CFD; it may replace the conventional supercomputers in the near future. Hu and Jackson [4], and Chriske and Boguez [2], have made the performance study for a two dimensional source panel method calculation, and the study shows that the parallel code achieved a high performance. The potential of MPP computers is being realized. Computational Fluid Dynamics (CFD) is being enhanced immensely by the advent of MPP computers. Various CFD problems are being solved by MPP computers in a more efficient manner. Thus, the study of the

performance of MPP computers on CFD problems has become a very important issue.

In the realm of Computational Fluid Dynamics, the finite difference method (FDM) and the finite volume method (FVM) are major methods of solving incompressible and compressible flows, including transonic flow problems. The FDM and FVM for the Navier-Stokes equations, though successful, are expensive to implement. However, the Integral Equation Method(IEM) applied to the potential equation formulation for incompressible and some compressible flows, including transonic flows, represents an efficient alternative to FDM and FVM.

The IEM, based on Green's theorem, represents the solution in terms of integrals over the computational domain and boundary of the domain. The IEM computation for typical aerodynamics problems involves evaluating a large number of integrals, which is expensive to evaluate on serial-architecture supercomputers. In contrast, these integral computations may be done less expensively in a massively parallel computing environment. Therefore, there is a need to study its performance.

The IEM for linearized subsonic and supersonic flow computations has been in use since the 1960's and has become an indispensable tool in aerodynamic analysis and design. A review of the method is given by Kandil and Yates [6]. Since the 1970's, several numerical schemes implementing the IEM for nonlinear compressible including transonic flows have been developed by some investigators [5-6].

For example, Hu [5] developed a numerical scheme using the IEM in solving the full-potential equation for three dimensional compressible including transonic flows. The numerical scheme is implemented on the conventional supercomputer Cray-YMP.

The objective of this work is to implement, including the necessary modifications, the numerical scheme of the IEM developed by Hu [5] in a massively parallel

computing environment. A comparative study on the performance of the parallel code and the serial code for three dimensional calculations is made.

Chapter 2 presents the physical and mathematical foundation of the problem, along with the integral equation solution and its associated numerical algorithm. Chapter 3 discusses the CM-5 architecture, parallel CM-FORTRAN computer language, and implementation of parallel computation of the IEM. The performance study is made in Chapter 4. Finally, the concluding remarks on this investigation are given in Chapter 5.

CHAPTER 2

MATHEMATICAL FORMULATION

2.1 The Physical Problem

The physical problem being studied is the flow around three dimensional aerodynamic configurations. Incompressible and compressible flows are considered. In order to study flows, it is necessary to study the laws of conservation of mass, momentum and energy, which lead to the fundamental governing equations for fluid flows. The following derivation of the fundamental governing equations are given by Anderson [1]. The derivation of the full-potential equation is based on Hu [5] but is modified to meet the specifications of the problem solved.

2.2 Fundamental Equations of Fluid Flows

There are three fundamental equations for fluid flows, the continuity equation, the momentum equation and the energy equation. Each equation has its physical interpretation.

2.2.1 The Continuity Equation

The physical principle of the continuity equation can be stated that the mass

can be neither created nor destroyed. The principle is expressed mathematically as

$$\frac{\partial}{\partial t} \int_V \rho dV + \int_S \rho \vec{V} \cdot d\vec{S} = 0 \quad (2.1)$$

where the first term is the time rate of increase of mass inside control volume, V ; the second term represents the flow across the boundary, S , of the control volume, V ; \vec{V} is the flow field velocity vector, ρ the density and t the time. Applying the divergence theorem, Equation (2.1) becomes

$$\int_V \left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) \right] dV = 0 \quad (2.2)$$

For arbitrary control volume, V , Equation (2.2) implies that

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0 \quad (2.3)$$

which is the continuity equation in differential equation form.

2.2.2 The Momentum Equation

The physical principle of the momentum equation is that the force equals time rate of change of momentum. The equation in integral form is

$$\frac{\partial}{\partial t} \int_V \rho \vec{V} dV + \int_S (\rho \vec{V} \cdot d\vec{S}) \vec{V} = - \int_S p d\vec{S} + \int_V \rho \vec{f}_b dV + \vec{F}_v \quad (2.4)$$

where p is the pressure; $d\vec{S}$ is the normal vector of the surface elements. $\frac{\partial}{\partial t} \int_V \rho \vec{V} dV$ is the time rate of change of momentum contained at any instant inside the control volume due to flow fluctuations. The net flow of momentum out of the control volume through surface S is the summation of the above elemental contributions and is expressed as $\int_S (\rho \vec{V} \cdot d\vec{S}) \vec{V}$. $-\int_S p d\vec{S}$ is the pressure force acting on the

control volume. $\int_V \rho \vec{f}_b dV$ is the body force and \vec{F}_v is the viscous force. Applying the divergence theorem to surface integral terms of Equation (2.4), the momentum equation becomes

$$\int_V \left[\frac{\partial(\rho \vec{V})}{\partial t} + \nabla \cdot (\rho \vec{V} \vec{V}) + \nabla p - \rho \vec{f}_b - \vec{f}_v \right] dV = 0 \quad (2.5)$$

where \vec{f}_v represents the viscous force term. Since the control volume V is arbitrarily chosen, Equation (2.5) gives

$$\frac{\partial \rho \vec{V}}{\partial t} + \nabla \cdot \rho \vec{V} \vec{V} + \nabla p - \rho \vec{f}_b - \vec{f}_v = 0 \quad (2.6)$$

which is the momentum equation in differential equation form.

2.2.3 The Energy Equation

The physical principle of the energy equation is that energy can be neither created nor destroyed; it can only change form. The energy equation is expressed as

$$\int_V \dot{q} \rho dV + \dot{Q}_v - \int_S p \vec{V} \cdot d\vec{S} + \int_V \rho (\vec{f} \cdot \vec{V}) dV + \dot{W}_v = \frac{\partial}{\partial t} \int_V \rho \left(e + \frac{V^2}{2} \right) dV + \int_S \rho \left(e + \frac{V^2}{2} \right) \vec{V} \cdot d\vec{S} \quad (2.7)$$

where \dot{W}_v is work due to viscous effects. \dot{Q}_v is due to viscous effects. \dot{q} is the volumetric rate of heat addition per unit mass. \dot{Q}' is the total rate of heat addition to the control volume due to viscous effects. \dot{W}' is the total work done by viscous effects on the control surface S . t is the time.

The partial differential form of Equation (2.7) is given by

$$\frac{\partial}{\partial t} \left[\rho \left(e + \frac{V^2}{2} \right) + \nabla \cdot \left[\rho \left(e + \frac{V^2}{2} \right) \vec{V} \right] \right] = \rho \dot{q} - \nabla \cdot (p \vec{V}) + \rho (\vec{f}_b \cdot \vec{V}) + \dot{Q}' + \dot{W}' \quad (2.8)$$

2.3 Full-Potential Equation

The Equations (2.3), (2.6) and (2.8) are combined to form the Navier-Stokes equations in the computational fluid dynamics, although in the theoretical fluid mechanics, Equation (2.6) alone is called the Navier-Stokes equations. They represent the most general governing equations for fluid flows. However, for some flows, there are simplified governing equations. Flows in which vorticity is zero are known as irrotational flows. Fluid elements of irrotational flow field have no angular velocity. The mathematical relation representing irrotational flow is

$$\nabla \times \vec{V} = 0 \quad (2.9)$$

which means that the curl of velocity vector field \vec{V} is zero. Comparing Equation (2.9) with the vector identity, $\nabla \times \nabla \Phi = 0$, it is found that

$$\vec{V} = \nabla \Phi \quad (2.10)$$

This implies that for irrotational flows, there exists a scalar function Φ , the velocity potential. Inviscid flows are flows in which viscous effects are negligible. Compressible flows as opposed to incompressible flows are flows in which the density ρ varies.

For unsteady inviscid compressible flows with negligible body forces, the continuity and momentum equations, Equations (2.3) and (2.6) reduces to

$$\frac{\partial \rho}{\partial t} + \vec{V} \cdot \nabla \rho + \rho \nabla \cdot \vec{V} = 0 \quad (2.11)$$

and

$$\frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \nabla \vec{V} + \frac{1}{\rho} \nabla p = 0 \quad (2.12)$$

respectively.

Using the identity,

$$\vec{V} \cdot \nabla \vec{V} = \nabla \frac{V^2}{2} - \vec{V} \times (\nabla \times \vec{V}) \quad (2.13)$$

Equation (2.12) becomes

$$\frac{\partial \vec{V}}{\partial t} + \nabla \left(\frac{V^2}{2} \right) - \vec{V} \times (\nabla \times \vec{V}) + \frac{1}{\rho} \nabla p = 0 \quad (2.14)$$

For steady, irrotational flows, $\frac{\partial}{\partial t} = 0$ and $\nabla \times \vec{V} = 0$. Equation (2.14) reduces to

$$\nabla \left(\frac{V^2}{2} \right) + \frac{1}{\rho} \nabla p = 0 \quad (2.15)$$

Using the velocity potential, Φ , with

$$\vec{V} = \nabla \Phi$$

then Equation (2.15) becomes

$$\frac{\nabla (\nabla \Phi)^2}{2} + \frac{1}{\rho} \nabla p = 0 \quad (2.16)$$

Integrating Equation (2.16) with respect to space yields

$$\frac{(\nabla \Phi)^2}{2} + \int \frac{dp}{\rho} = g \quad (2.17)$$

For barotropic fluid, there is the relation

$$\int \frac{dp}{\rho} = \frac{a^2}{\kappa - 1} \quad (2.18)$$

where a is the speed of sound and κ is the gas specific heat ratio. If the fluid is at rest at infinity, then it follows that

$$g = \frac{a_\infty^2}{\kappa - 1} \quad (2.19)$$

where subscript ∞ refers to the infinity condition. Substituting Equation (2.18) and (2.19) into (2.17) yields

$$\frac{(\nabla\Phi)^2}{2} + \frac{a^2}{\kappa - 1} = \frac{a_\infty^2}{\kappa - 1} \quad (2.20)$$

Assuming that the flow is isentropic and the isentropic relation, which is another form of the energy equation,

$$\frac{\rho}{\rho_\infty} = \left(\frac{a^2}{a_\infty^2} \right)^{\frac{1}{\kappa-1}} \quad (2.21)$$

is used into Equation (2.20) to get

$$\frac{\rho}{\rho_\infty} = \left[1 - \frac{(\kappa - 1)}{2a_\infty^2} (\nabla\Phi)^2 \right]^{\frac{1}{\kappa-1}} \quad (2.22)$$

By defining the Mach number M_∞ as ratio of V_∞ and a , Equation (2.22) takes the form

$$\frac{\rho}{\rho_\infty} = \left[1 - \frac{(\kappa - 1)}{2} \left[\frac{M_\infty}{V_\infty} \right]^2 (\nabla\Phi)^2 \right]^{\frac{1}{\kappa-1}} \quad (2.23)$$

For steady, irrotational flows, the continuity equation, Equation (2.11), reduces to

$$\nabla\Phi \cdot \nabla\rho + \rho \nabla \cdot \nabla\Phi = 0 \quad (2.24)$$

which can be rewritten in the following form

$$\nabla^2\Phi = -\frac{1}{\rho} \nabla\rho \cdot \nabla\Phi \quad (2.25)$$

After introducing the characteristic parameters of V_∞ , ρ_∞ and the wing root chord length c , Equations (2.25) and (2.23) take the dimensionless form as follows

$$\nabla^2\Phi = G \quad (2.26)$$

with

$$G = -\frac{1}{\rho} \nabla\rho \cdot \nabla\Phi \quad (2.27)$$

and

$$\rho = \left[1 - \frac{(\kappa - 1)}{2} M_\infty^2 (\nabla \Phi)^2\right]^{\frac{1}{\kappa - 1}} \quad (2.28)$$

Equation (2.26) along with Equations (2.27) and (2.28) is the full-potential equation for inviscid compressible flows. For transonic flows, Equation (2.26) along with Equations (2.27) and (2.28) is a mixed elliptic-hyperbolic partial differential equation. It should be noted that equation (2.26) is written in the poisson's equation form, where the compressibility, G , is recognized as an inhomogeneity instead of a nonlinearity. In order to study the mixed nature, one may investigate the Transonic Small Disturbance (TSD) equation, which is obtained from the full-potential equation under the small disturbance assumption. The TSD equation can be written as

$$[1 - M^2(x, y, z)] \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = 0 \quad (2.29)$$

From Equation (2.29) when the local Mach number, $M(x, y, z)$ is larger than one, the equation is a hyperbolic partial differential equation. When $M(x, y, z)$ is less than one, the equation is a elliptic partial differential equation.

2.4 Boundary Conditions

The boundary conditions are surface no-penetration condition, Kutta condition, infinity condition, and wake kinematic and dynamic conditions. They are described as follows:

$$\vec{V} \cdot \vec{n}_g = 0 \quad \text{on} \quad g(\vec{r}) = 0 \quad (2.30)$$

$$\Delta C_p \big|_{sp} = 0 \quad (2.31)$$

$$\nabla \Phi \rightarrow 0 \quad \text{away from} \quad q(\vec{r}) = 0 \quad \text{and} \quad w(\vec{r}) = 0 \quad (2.32)$$

$$\vec{V} \cdot \vec{n}_w = 0 \quad \text{on} \quad w(\vec{r}) = 0 \quad (2.33)$$

and

$$\Delta C_p = 0 \quad \text{on} \quad w(\vec{r}) = 0 \quad (2.34)$$

where \vec{n}_g is the unit normal vector of the wing surface, $g(\vec{r}) = 0$; C_p is the surface pressure coefficient; the subscript sp refers to the edges of separation; and $w(\vec{r}) = 0$ is wake surface(s). For nonlifting symmetric flow where the wake surface can be neglected, the required boundary conditions are surface no-penetration and infinity condition given by Equations (2.30) and (2.31), respectively.

2.5 Integral Equation Solution

By using the Green's theorem, the integral equation solution in terms of the velocity field for nonlifting flows is given by

$$\begin{aligned} \vec{V}(x, y, z) = & \vec{V}_\infty \\ & - \frac{1}{4\pi} \int \int_g \frac{q_g(\xi, \eta, \zeta)}{d^2} \vec{e}_d ds(\xi, \eta, \zeta) \\ & + \frac{1}{4\pi} \int \int \int_v \frac{G(\xi, \eta, \zeta)}{d^2} \vec{e}_d d\xi d\eta d\zeta \\ & - \frac{1}{4\pi} \int \int_S \frac{q_s(\xi, \eta, \zeta)}{d^2} \vec{e}_d ds(\xi, \eta, \zeta) \end{aligned} \quad (2.35)$$

where \vec{V}_∞ is the free-stream velocity; q is the surface source distribution; the subscript, S , refers to the shock surface; ds is the infinitesimal surface area; the vector \vec{d} is given by $\vec{d} = (x - \xi)\vec{i} + (y - \eta)\vec{j} + (z - \zeta)\vec{k}$; and \vec{e}_d is defined by $\vec{e}_d = \vec{d}/|\vec{d}|$. It can be seen that the infinity condition is automatically satisfied by the integral

equation solution.

2.6 Discretization

To solve the problem using field-panel method, Equation (2.35) must be discretized. The discretization in terms of field-panels is as follows

$$\begin{aligned}
\vec{V}(x, y, z) = & \vec{V}_\infty \\
& - \frac{1}{4\pi} \sum_{i=1}^{LG} \sum_{k=1}^{NG} q_{g_{i,k}} \int \int_{g_{i,k}} \frac{1}{d^2} \vec{e}_d ds(\xi, \eta, \zeta) \\
& + \frac{1}{4\pi} \sum_{i=1}^{LV} \sum_{j=1}^{MV} \sum_{k=1}^{NV} G_{i,j,k} \int \int \int_{\forall_{i,j,k}} \frac{1}{d^2} \vec{e}_d d\xi d\eta d\zeta \\
& + \frac{1}{4\pi} \sum_{j=1}^{MS} \sum_{k=1}^{NS} q_{s_{j,k}} \int \int_{S_{j,k}} \frac{1}{d^2} \vec{e}_d ds(\xi, \eta, \zeta)
\end{aligned} \tag{2.36}$$

where the indices, i , j and k refer to the surface and field panels; $LG \times NG$ is the total number of wing surface panels; $LV \times MV \times NV$ is the total number of field panels; and $MS \times NS$ is the total number of shock surface panels. The constantly distributed surface and volume sources are used.

2.7 Numerical Scheme

2.7.1 Panel Method for Incompressible Flows

The standard panel method is used to solve the incompressible flow problems. In Equation (2.36), $G = 0$ for incompressible flows, and hence $q_s = 0$. Equation (2.36) reduces to

$$\vec{V}(x, y, z) = \vec{V}_\infty - \frac{1}{4\pi} \sum_{i=1}^{LG} \sum_{k=1}^{NG} q_{g_{i,k}} \int \int_{g_{i,k}} \frac{1}{d^2} \vec{e}_d ds(\xi, \eta, \zeta) \tag{2.37}$$

After applying the wing surface zero-normal-velocity boundary condition at each control point (CP) of all panels,

$$\vec{V}(x, y, z) \cdot \vec{n} = 0 \quad (2.38)$$

a system of equation is obtained,

$$[A]\{q\} = \{B\} \quad (2.39)$$

where $[A]$ is $N \times N$ aerodynamic influence coefficient matrix, and $N = LG \times NG$; $\{q\}$ is a $N \times 1$ unknown vector matrix containing q_j for $j = 1$ to N ; and $\{B\}$ is a $N \times 1$ known vector matrix which is contributed from \vec{V}_∞ . The solution procedure of the problem using source panel involves three major steps: Step 1 - evaluation of integrals for N^2 times to construct matrices $[A]$ and also $\{B\}$; Step 2 - solving the resulting dense linear system of Equation (2.39); and Step 3 - post-processing of aerodynamic calculations.

Step 1 involves evaluating a large number of integrals. The total number of integrals can be very large for aerodynamic problems. It can be in the order of 10^8 if $LG \times NG = 100 \times 100$. An important feature of step 1 is that the calculation for each (x, y, z) and each (ξ, η, ζ) can be performed simultaneously for all (x, y, z) and all (ξ, η, ζ) . This feature of panel method calculation leads itself in a natural way for processing data in a parallel computing environment.

2.7.2 Field Panel Method for Compressible Flows

The field Panel method is used to solve the problem. The scheme developed by Hu [5] in serial FORTRAN code is inherently used in developing the parallel CM FORTRAN version.

The solution procedure is as follows:

Step 1 - Standard Panel Method calculations.

Step 2 - Computation of the Initial Value of the Compressibility: Unlike the incompressible case where the density ρ is constant, and the compressibility, G , is zero, G is updated by using an iterative method. A central difference scheme is used to calculate the derivative of ρ .

Step 3 - Enforcing the Boundary Conditions: The boundary conditions are enforced in order to develop the appropriate matrices, where the compressibility, G , obtained in Step 2 and the source distribution obtained in Step 1 are used. The solution gives q_g distributions.

Step 4 - Calculation of the Surface Pressure Coefficients: Once G and the source distribution q_g are obtained, they are used in Equation (2.36) to calculate the velocity at each control point. The surface pressure coefficients are then calculated. The pressure coefficient is defined by

$$C_p \equiv \frac{2}{\kappa M_\infty^2} \left(\frac{p}{p_\infty} - 1 \right) \quad (2.40)$$

where p and p_∞ are all dimensional quantities. By introducing the isentropic flow relation,

$$\frac{p}{p_\infty} = \left(\frac{\rho}{\rho_\infty} \right)^\kappa \quad (2.41)$$

and writing $\frac{\rho}{\rho_\infty}$ as the dimensionless density, ρ , Equation (2.41) becomes

$$C_p = \frac{2}{\kappa M_\infty^2} (\rho^\kappa - 1) \quad (2.42)$$

Equation (2.42) is used to compute pressure coefficients at each control point of the wing surface.

Step 5 - Steps 2-4 are repeated until convergence.

2.8 Significance of Computing In a Parallel Environment

The IEM computations for typical aerodynamics problems involves evaluating a large number of integrals over the computational domain and the boundaries of the domain as indicated in Equation (2.36). The computation of this large number of integrals on serial-architecture supercomputers is very computationally expensive. On the other hand, it is noticed that all these integral computations can be computed simultaneously in a massively parallel processing environment. Therefore, it is imperative to study the computational performance of the IEM on a MPP environment. An overview of the MPP and CM Fortran is given in the following chapter.

CHAPTER 3

PARALLEL IMPLEMENTATION

3.1 Connection Machine - 5

The connection machine is a MPP computer that is the brainchild of W. Daniel Hillis, Marvin L. Minsky and Scott E. Falman, collectively producing the first prototype in 1985 [3]. In MPP computers, many small processors are made to work simultaneously, each accompanied by small memory. Parallel processing allows memory capacity and processing capacity to be utilized yielding high efficiency. Each processor is much less powerful and less expensive to produce than a typical conventional supercomputer processor, but in tandem they can achieve an extremely high performance which conventional supercomputers can not achieve [3]. Since each processor is bearing down on the same problem there has to be a means by which they communicate. The user has the option to use the SIMD(Single Instruction Multiple Data) mode of data parallel computing or the MIMD(Multiple Instruction Multiple Data) mode of message passing computing. The present study of performance uses the SIMD mode where the communication is solely controlled by the computer implicitly as opposed to MIMD mode where the user explicitly controls necessary communication.

The Connection Machine CM-5 system is a scalable distributed-memory mul-

tiprocessor system [8]. The architecture of the CM-5 is optimized for parallel processing of large, complex problems [8]. The system is designed to operate on large amounts of data. The major hardware elements of the system include Front-End computers to provide developing and execution environments and a parallel processing unit, which consists of multiple nodes, to execute parallel operations [8]. The basic computational unit is the compute node. Compute nodes work in parallel. Each node has its own memory. The processing nodes are supervised by a control processor; it broadcasts blocks of instructions to the parallel processing nodes and then initiates execution. The CM-5 parallel processing nodes are divided into groups, known as partitions. Every control processor and parallel processing node in the CM-5 is connected to two scalable interprocessor communication networks, designed to give low latency combined with high bandwidth in any possible configuration a user may wish to apply to a problem [8].

Each compute node contains 1 sparc processor, four vector units(vector length = 16) with 32 Megabytes(MB) of memory. A theoretical peak performance is 128 MFLOPS per node for a total of 16 GFLOPS no the 128 node CM-5 [8].

3.2 CM-Fortran

The software consists of CM-FORTRAN which is a high performance FORTRAN language, the CMSSL (Connection Machine Scientific Software Library), a debugging software package, Prism, and a host of others. The CM-FORTRAN language is an implementation of FORTRAN-77 supplemented with array-processing extensions from the standard FORTRAN-90 [9]. These array-processing features map naturally onto SIMD data parallel architecture of the CM-5 system, since the CM-FORTRAN allows array elements to be evaluated simultaneously. The

most important difference between CM-FORTRAN and standard FORTRAN is the treatment of entire arrays as objects, thus explicit indexing in CM-FORTRAN is not always necessary. For example, it is not necessary to write DO-LOOPS or other such control constructs to have the operation repeated for each element of an array.

3.3 Parallel Implementation

A three dimensional, steady, incompressible and compressible, source field panel method is converted to a parallel CM FORTRAN program. Before manual conversion, the CMAX translator is used to partially convert some of the serial FORTRAN code into parallel CM-FORTRAN code under the data parallel (SIMD) programming mode. Since CMAX is relatively new and not very well developed yet a lot of the conversions made by CMAX have to be fine tuned. Hence, most of the conversion is done manually.

Prior to the description of the code conversion, it is necessary to describe some properties of CM-FORTRAN. Arrays in CM FORTRAN come in the following two forms, Front End(FE) arrays and Connection Machine(CM) arrays. Front-End Arrays are for standard FORTRAN operations and are stored on the Partition Manager and are called Front End (FE) arrays. CM arrays are for parallel FORTRAN operations and are stored across the local memories of the processing nodes [9]. While using FE and CM arrays one should avoid using an array as both an array object and a subscripted array. An array used as an array object always resides on the parallel processing nodes and hence has a CM home. If a CM array is used as subscripted array (FORTRAN), the system moves the CM array to the FE, one element at a time, to perform the sequential operations. This transfer of CM arrays

to FE arrays degrades the performance [9]. Intrinsic functions and other attributes of CM-FORTRAN are listed along with examples in the following sections.

3.3.1 Parallelization of Incompressible Flow Case

Step 1 of the incompressible source panel method is coded in subroutine VELWING.f, which constructs the linear dense matrices $[A]$ and $\{B\}$. The serial FORTRAN subroutine VELWING.f is converted to fully parallelized code. The original subroutine VELWING.f is partially listed in Figure 1a and its parallel counterpart VELWING.fcm in Figure 1b.

In subroutine VELWING.f there are two doubly nested DO-LOOPS, nested IF statements and a call to subroutine VWS.f, which is called from the inner doubly nested DO-LOOPS. Parallel WHERE assignments are implemented to replace the serial FORTRAN IF command. Instead of calling subroutine VWS.f, as in the serial version, VELWING.fcm replaces the call to VWS.f with a fully parallelized version of the VWS.f subroutine. All four DO-LOOPS are efficiently replaced by the intrinsic SPREAD functions. The FORALL command is used to manipulate the four dimensional arrays developed by implementing the SPREAD function. A brief overview of each CM-5 attributes and a few examples are given below.

3.3.1.1 Attributes of the SPREAD Function

The SPREAD function is a transformational function. It broadcasts copies of a source array(or scalar) along a dimension (as in forming a book from copies of single page), yielding an array of rank one greater than the source, with values replicated from the source [9].

Essentially the spread function is used to replace the nested DO-LOOPS by transforming all arrays within the two outer loops and the two inner loops into four dimensional arrays. For the smaller problem tested where $LG \times NG = 24 \times 12$,

DO-LOOP 1 goes from 1 to 24 and DO-LOOP 2 goes from 1 to 12. DO-LOOP 3 goes from 1 to 24 and DO-LOOP 4 goes from 1 to 6. The arrays formulated in the outer loops are copied along the third and fourth dimension. The arrays formulated in the inner loops are copied along the first dimension. The arrays are spread in this fashion to ensure correspondence. That is to say when arrays formulated in the outer loops 1 and 2 are spread to four dimensional arrays, and when arrays formulated within the inner loops 3 and 4 which are spread to four dimensional arrays, computations involving arrays originating from the outer loops and the inner loops will correspond correctly. Now all computations can be done simultaneously in four dimensions instead of serially by using time consuming nested DO-LOOPS. To see how the SPREAD function is applied, one may look at array $xcm4(24, 12, 24, 6)$ in Figure 1b. In the original code listed in Figure 1a, it is calculated serially within the DO-LOOPS 1 and 2 as scalar xc . In the parallel version $xc(nr, nc)$ is renamed $xcm(nr, nc)$. Its elements are calculated simultaneously by using the feature of CM-FORTRAN that simply lets you calculate the array instantaneously. Hence, using $xcm(:, nr, : nc) = \dots$ will compute and store the all value of $xcm(1, 1), \dots, xcm(24, 12)$ in array xcm simultaneously.

Since xc is located within the DO-LOOPS 1 and 2, but not the two inner DO- LOOPS 3 and 4 of VELWING.f, its values in the parallel version are initially located in the parallel array $xcm(24, 12)$. Then they are copied along the third dimension via the SPREAD function. Thus, $xcm3$ contains 24 copies of the values in xcm , where $xcm3$ is the three dimensional array $xcm3(24, 12, 24)$. Likewise, $xcm3$ is copied along the fourth dimension. It contains 6 copies of the values of $xcm3$. $xcm4$ is the four dimensional array $xcm4(24, 12, 24, 6)$.

The scalar xf calculated serially within the inner most DO-LOOPS 11 and 12 of the origin version can be calculated simultaneously in the parallel version.

In the parallel version it is initially the array $xfm(24,6)$. Then via spreading it along the 1st dimension and making 12 copies of xfm along the first dimension, it becomes $xfm3$ which stores the values in $xfm3(12,24,6)$. Then 24 copies of $xfm3$ is spread along the first dimension of $xfm4$ yielding $xfm4(24,12,24,6)$. Now these four dimensional arrays are manipulated as complete objects instead of being manipulated element by element.

3.3.1.2 Attributes of The FORALL Statement

The FORALL statement is an elemental array assignment statement used to specify an array assignment in terms of array elements of array sections. The FORALL statement effectively describes a collection of assignments to be executed elementally [9].

The FORALL statement used in VELWING.fcm and also VELWING2.fcm predominantly calculates in parallel the arrays and scalars that in VWS.f subroutine. It assigns elements to the necessary arrays. For example,

$z1(is, js, ir, jr)$ where, $is = 1$ to 24, $js = 1$ to 12, $ir = 1$ to 24 and $jr = 1$ to 6.

3.3.1.3 Attributes of The WHERE Statement

The WHERE construct qualifies the evaluation of expressions and assignments of values in several array assignment statements [9]. It is very similar to the well known IF construct of standard FORTRAN. Unlike the IF statement it requires everything within the WHERE construct to be an array of the same dimension of the masked expression. For example referring to Figure 3b the WHERE construct

is

```
WHERE(distm4.lt.farfd)

WHERE(distm4.lt.0.0001)

ELSEWHERE

ENDWHERE

ENDWHERE.
```

where *distm4* is the array used in the mask statements. Since it is a four dimensional array all array assignments and evaluations within the expression have to be the same dimension of array *distm4*(24,12,24,6).

All calculations needed to produce the desired results of subroutine VWS.f are placed in the parallel routines before the WHERE construct. The parallel version of VELWING.f drastically out-performs the original serial FORTRAN code. The results are discussed in Chapter 4.

3.3.1.4 Attributes of the Parallel Matrix System Solver

The CMSSL is a rapidly growing set of numerical routines that support computational applications while exploiting the massive parallelism of the Connection Machine system [10]. It includes dense and sparse matrix operations; routines for solving dense, banded, and sparse linear systems; eigensystem analysis routines[10]. Efficient use of the Connection Machine architecture is accomplished through a careful choice of data layout, efficient implementation of interprocessor data mo-

tion, and careful management of the local memory hierarchy and data paths in each processor [10].

The serial Gauss elimination routine is replaced by a call of subroutine Gauss elimination(LU decomposition) with partial pivoting. The system to be solved consists of dense CM array $[A]$ and CM array $\{B\}$. The CMSSL Gauss elimination routine permits the change of the blocking factor. The result of the performance as it relates to the change in the blocking factor is investigated and the results are presented in Chapter 4.

3.3.1.5 Parallelization of Post-Processing Calculations

Subroutine PRESS.f is used to calculate the pressure distribution over the surface of the wing and it is listed in Figure 2a. The parallel version is listed in Figure 2b. It is partially parallelized and this degrades the performance. Most of the subroutine is written in parallel, but most of the parallel computations occur inside of nested DO-LOOPS which are not parallel. This effects the performance immensely.

3.3.2 Parallelization of Compressible Flow Case

The serial FORTRAN program is based on the field panel method. The method is modified from the standard panel method so that the compressibility is accounted for by using an iterative scheme in which the density, and therefore G is updated during each iteration.

Subroutine GVICAL.f calculates the volume integrals, which is listed in Figure 3a. It consists of six nested DO-LOOPS. The inner DO-LOOPS, 11, 12, 13, each has conditional IF statements used to determine values for variables X, Y, Z . The

inner DO-LOOPS does the Near-field calculations. The outer nested three DO-LOOPS 6,7,8 are used to fix a receiver point at the lower right downstream corner of the control volume \forall . Also the surface and field velocities are saved for usage throughout the code.

GVICAL3.fcm is a fully parallelized version of GVICAL.f. It is listed in figure 3b. All calculation done in the DO-LOOPS are converted to FORALL statements or parallel array assignments. The IF statements assigning values to X, Y, Z , within the DO-LOOPS 11, 12, and 13 are replaced by array assignments $x100(20, 16, 18), x101(20, 16, 18)$, and

$y100(20, 16, 18), y101(20, 16, 18)$, and $z100(20, 16, 18), z101(20, 16, 18)$.

Other calculations within DO-LOOPS 11, 12, and 13 of GVICAL.f are replaced by parallel array assignment statements. The IF statement using the masks

$abs(xff), abs(yff)$ and

$abs(zff)$ is replaced by nested WHERE statements with arrays

$xff100, yff100$, and $zff100$ replacing

xff, yff , and zff respectively.

Scalars xs, ys , and zs in subroutine GVICAL.f are replaced by arrays

$xs100(20, 16, 18), ys100(20, 16, 18)$ and

$zs100(20, 16, 18)$, respectively, which are all formulated by using the FORALL command. Although the code is expanded for 82 lines of FORTRAN code to 388 lines of CM FORTRAN code, the parallel version is the best version for application

on MPP computers. Also it out-performs the original serial code.

Subroutine VELFDG.f listed in Figure 4a takes the summation of all of the integrals evaluated by subroutine GVICAL.f. Subroutine VELFDG.f is converted to the fully parallel version VELFDG.fcm, which is listed in Figure 4b. This subroutine accounts from the effect of compressibility. DO-LOOPS 6, 7, and 8 of subroutine VELFDG.f are replaced by FORALL statements in subroutine VELFDG.fcm.

CHAPTER 4

PERFORMANCE ANALYSIS

Studying each phase of the CM-FORTRAN version of the source field panel method shows that the CM-5 has its advantages and disadvantages. The serial and parallel codes for the standard panel method and the field-panel method are executed on Cray-YMP with a single processor and CM-5 with 32, 64 and 128 nodes, respectively. The performance on Cray-YMP supercomputer with a single processor provides the basis for comparison. The computational performance is obtained using Cray-YMP's PERFTRACE utility. The CPU execution (CM-busy) time on the CM-5 is obtained via CM TIMER routines.

Identical pressure distribution results are obtained using both serial FORTRAN code and parallel CM-FORTRAN code for the incompressible flow case. This part of results is published by Logan and Hu [7]. Results of the parallel code for the compressible flow case are different from the results accrued on the Cray-YMP. Performance of the code using 1 iteration and 6 iterations is presented for the compressible flow case.

4.1 Performance of Incompressible Flow Computation

Results relating to how CM-5 performance compares to the Cray-YMP are presented in Tables 1 through 4. Graphs 1 thru 4 are developed from results listed

in these corresponding tables. Partially parallelized code as opposed to fully parallelized code has a significant effect on performance. This effect and other pertinent details are presented below.

Tables 1 thru 3 gives CPU time required for each step of the incompressible flow source panel method as mentioned. It lists results obtained for the smaller problem where $N = 24 \times 12$, 288 source panels and the larger problem where $N = 48 \times 24$, 1152 source panels.

Table 1 lists CPU time used to construct the matrices. This segment of code is fully parallelized. It is found that CM-5 performs better than Cray-YMP on both smaller and larger problems. Scaling the CM-5 up to 64 nodes and then to 128 nodes results in even faster CPU time. Comparing the Cray-YMP to the CM-5 as the problem size increases, it is seen that the Cray-YMP requires approximately twenty times as much CPU time to complete the larger problem than it requires to complete the smaller problem. However, CM-5 requires at most ten times as much CPU time to complete the larger problem as it requires to complete the smaller one. Since the CPU time decreases considerably as the number of nodes increases, this implies that the code is not only fully parallelized but it is also efficiently implemented on the parallel processing unit.

Table 2 lists CPU time used to solve the matrices. Gaussian elimination is used to solve the matrices on the conventional Cray-YMP. A CM-FORTRAN Gaussian elimination library routine is used to solve the matrices on the CM-5. Cray-YMP performs faster than CM-5 for the smaller problem requiring less than half the time it takes CM-5 to solve the problem using 128 nodes. However, CM-5 performs faster than Cray-YMP for the larger problem. Also as the problem size increases

Cray-YMP requires fifty-eight times as much CPU time to solve the larger problem than it does to solve the smaller problem. However, CM-5 requires at most six times as much CPU time to solve the larger problem than it does to solve the smaller problem. CPU time does not consistently decrease as the number of nodes used increases. This may be due to CM-5 communication overhead.

Table 3 lists CPU time used for post-processing. This part of the CM-FORTRAN code is partially parallelized. Partially parallelizing the code has a significant effect on the performance of CM-5. Thus, Cray-YMP performs faster than CM-5 for the smaller and larger problems. It completes the task in approximately one-tenth the time it takes CM-5 to complete the task. However, as the problem size increases Cray-YMP requires approximately seventeen times as much CPU time to solve the larger problem than it does to solve the smaller problem. CM-5 requires close to four and one-half times as much CPU time to solve the larger problem than it does to complete the smaller problem.

Table 4 lists total CPU time required to solve the incompressible flow part via the source panel method. Cray-YMP performs faster than CM-5 on the smaller problem. It requires close to one-third times the CPU time it takes CM-5 to solve the smaller problem. CM-5 performance may be attributed to the fact that some of the code is partially parallelized. Its performance may also be attributed to communication overhead. However, CM-5 outperforms Cray-YMP on the larger problem. Cray-YMP requires thirty times as much CPU time to solve the larger problem than the smaller problem, whereas CM-5 only requires five and one-third times as much CPU time to solve the larger problem than the smaller problem. Overall, CM-5 outperforms Cray-YMP in implementing the source panel method

on incompressible flow problems.

4.2 Performance of Compressible Flow Computations

Results for the compressible flow case obtained from the field panel method are obtained. The computational results are not identical to the results obtained on the Cray-YMP, and some investigations are being made. The performance of the CM-5 CM-FORTRAN version of the code is compared with the Cray-YMP serial FORTRAN version. The code is executed using 1 iteration and again using 6 iterations. The problem sizes increases from $N = 24 \times 12$, 288 source panels to $N = 48 \times 24$, 1152 source panels. The results are listed in Tables 5 and 6. Graphs 5 and 6 re-represents results listed in Table 6.

Table 5 lists CPU time for evaluating volume integrals. CM-5 outperforms Cray-YMP in completing this task. The CM-5 subroutine that completes this task is fully parallelized. Also as the number of nodes increases CPU time consistently decreases by a considerable amount. The performance of Cray-YMP diminishes on the larger problem. However, CM-5 attains the same CPU time it attains for the larger problem as it does for the smaller problem. Overall CM-5 outperforms Cray-YMP on both problems. More importantly, it does not require additional CPU time to solve the larger problem like Cray-YMP does. CM-5 good performance can be attributed to the fact that the code is fully parallelized and it is efficiently implemented on the parallel processing unit.

Table 6 lists total CPU time for 6 iterations. Cray-YMP performs better than CM-5 on both the smaller and larger problem. This can be attributed to the fact that the CM-5 version uses subroutines that are partially parallelized. This causes communication overhead to be very high. However, it is noticed that as the problem

size increases Cray-YMP, CPU time increases by a factor of four and four-fifths, whereas CM-5 CPU time only increases by a factor of one and one-third. This implies that CM-5 may be the best choice for large scale problems, since CPU time increases by a small margin compared to Cray-YMP.

CHAPTER 5

CONCLUDING REMARKS

The study of an integral equation method for a three dimensional aerodynamics problem is made in a massively parallel processing (MPP) environment. Serial FORTRAN code is converted into parallel CM-FORTRAN code and a comparative study is made to determine how well the CM-5 MPP computer performs when compared to the conventional Cray-YMP supercomputer. Since CM-5 is intended for use on large scale problems, two different problem sizes are tested to see how the size of the problem effects the performance.

From this investigation, the following concluding remarks can be made: (1) Whenever the parallel code is fully parallelized, CM-5 out-performs Cray-YMP; (2) For the small problem, a partially parallel code is not appropriate for CM-5; and (3) When the size of the problem increases, the performance of CM-5 increases. For further study, it is suggested that code should be fully parallelized in all subroutines in order to achieve an overall high performance in a MPP environment.

```

SUBROUTINE VELWING(IVELCT,IWG,IG,JG,KG)
COMMON/BLK01/X(25,13),Y(25,13),Z(25,13)
.....
DO 1 JS=1,NC
JS1=JS+1
DO 2 IS=1,NR
IS1=IS+1
X1=X(IS,JS)
.....
XC=(X1+X2+X3+X4)/4.0
.....
DO 11 JR=1,NC/2
JR1=JR+1
DO 12 IR=1,NR
.....
XF=0.25*(X(IR,JR1)+X(IR1,JR)+X(IR,JR)+X(IR1,JR1))
.....
DX=XF-XC
.....
DIST=SQRT(DX*DX+DY*DY+DZ*DZ)
IF(DIST.LT.FARFD) THEN
  IF(DIST.LT.0.0001) THEN
    UC=0.5*UNX(IS,JS)
    VC=0.5*UNY(IS,JS)
    WC=0.5*UNZ(IS,JS)
  ELSE
    CALL VWS(X1,X2,Z1,Z3,Y1,XF,YF,ZF,IS,JS,UC,VC,WC)
  END IF
ELSE
  AREAXZ=ABS((X2-X1)*(Z3-Z1))
  XYN=UNX(IS,JS)/UNY(IS,JS)
  ZYN=UNZ(IS,JS)/UNY(IS,JS)
  FACXZS=SQRT(1.0+XYN*XYN+ZYN*ZYN)
  AREAS=FACXZS*AREAXZ
  CONSTFF=OPI4*AREAS/(DIST*DIST*DIST)
  UC=CONSTFF*DX
  VC=CONSTFF*DY
  WC=CONSTFF*DZ
END IF
.....
NBA=(JS-1)*NR+IS
A(KEQ,NBA)=A(KEQ,NBA)-(UC*UNX(IR,JR)
+
+VC*UNY(IR,JR)
+
+WC*UNZ(IR,JR))
12 CONTINUE
11 CONTINUE
2 CONTINUE
1 CONTINUE
RETURN
END

```

Figure 1a. Serial Version - Constructing [A].

```

SUBROUTINE velwing(ivelct,iwg,ig,jg,kg)
include '/usr/cm/include/cm/CMF_defs.h'
COMMON/BLK01/X(25,13),Y(25,13),Z(25,13)
.....
real unxm3(24,12,24),vnxm3(24,12,24),wnxm3(24,12,24)
.....
unxm3 = spread(unx(:nr,:nc),dim=3,ncopies=24)
.....
xcm3 = spread(xcm,dim=3,ncopies=24)
.....
xcm4 =spread(xcm3,dim=4,ncopies=6)
.....
xfm3 = spread(xfm,dim=1,ncopies=12)
.....
xfm4 = spread(xfm3,dim=1,ncopies=24)
.....
dxm4 = xfm4 -xcm4
.....
dism4 = sqrt(dxm4*dxm4+dym4*dym4+dzm4*dzm4)
.....
where (dism4.lt.farfd)
  where (dism4 .LT. 0.0001)
    ucm4 = 0.5 * unxm4
    .....
  elsewhere
    xynm4 = unxm4/vnxm4
    zynm4 = wnxm4/vnxm4
    facxzsm4 = sqrt(1.0 + xynm4 * xynm4 + zynm4 * zynm4)
    ddxm4 = ddxm4
    ddzm4 = ddzm4
    fac4 = opi4*facxzsm4*abs(ddxm4*ddzm4)/6.0
    vwx4 = ffx11+ffx21+ffx31+ffx41+ .....
&    + ffx44+ffx15+ffx25+ffx35+ffx45
    .....
    ucm4 = vwx4*fac4
    .....
  endwhere
elsewhere
  zynm4 = wnxm4/vnxm4
  facxzsm4 = sqrt(1+xynm4*xynm4 + zynm4*zynm4)
  contm4 = opi4*areasm4/(dism4*dism4*dism4)
  vcm4 = contm4*dym4
endwhere
forall (jr=1:nc/2,ir=1:nr,js=1:nc,is=1:nr)
& a(ir+(jr-1)*nr,is+(js-1)*nr) = a(ir+(jr-1)*nr,is+(js-1)*nr)
& -((ucm4(is,js,ir,jr)
& *unxd4(is,js,ir,jr))+
& (vcm4(is,js,ir,jr)*vnxd4(is,js,ir,jr))+(wcm4(is,js,ir,jr)
& *wnxd4(is,js,ir,jr)))
return
end

```

Figure 1b. CM Parallel Version - Constructing [A].

```

SUBROUTINE PRESS(ITER,ICS,INC)
DIMENSION CPU(24,12,40)
COMMON/BLK01/X(25,13),Y(25,13),Z(25,13)
.....
DO 1 JR=1,NC/2
  JR1=JR+1
  DO 2 IR=(IUP-1)*NR/2+1,IUP*NR/2
    IR1=IR+1
    XF=0.25*(X(IR,JR1)+X(IR1,JR)+X(IR,JR)+X(IR1,JR1))
    .....
    CALL VELWING(1,2,1,1,1)
    OMTRX=BETAR*(ZF-ZP)-ALFAR*(YF-YP)
    .....
    UTRMS=E0X*HAM+OMTRX
    .....
    IF(INC.EQ.0) THEN
      UR=VCX-UTRMS
      .....
      UVWR2=UR*UR+VR*VR+WR*WR
      CPU(IR,JR,ITER)=1.0-UVWR2/HAM2
    ELSE
      CALL VELFDG(ITER,ICALREC,IR,JR,1,XF,YF,ZF,
&      UTT,VTT,WTT)
      UR=VCX+UTT-UTRMS
      .....
      UVWR2=UR*UR+VR*VR+WR*WR
      UVWTRM2=UTRMS*UTRMS+VTRMS*VTRMS+WTRMS*WTRMS
      .....
      RHO=(1.0+0.5*GM1*(-UVWR2+UVWTRM2-2.0*DPHIJR))
+      *(1.0/GM1)
      CPU(IR,JR,ITER)=2.0/GAMA/HAM2*(RHO**GAMA-1.0)
      END IF
      UVWR=SQRT(UVWR2)
      HAML=UVWR/(RHO** (0.5*GM1))
      END IF
2 CONTINUE
1 CONTINUE
RETURN
END

```

Figure 2a. Serial Version Post-processing.

```

SUBROUTINE press(iter,ics,inc)
DIMENSION CPU(24,12,40)
integer aaa,bbb,ccc
COMMON/BLK01/X(25,13),Y(25,13),Z(25,13)
.....
do jr = 1 , nc / 2
  jr1 = jr + 1
  do ir = nr/2+1,nr
    ir1 = ir + 1
    xf = 0.25 * (x(ir,jr1) + x(ir1,jr) + x(ir,jr) +
&    x(ir1,jr1))
    .....
    call velwing2(ccc,bbb,ccc,ccc,ccc)
    omtrx = betar * (zf - zp) - alfar * (yf - yp)
    .....
    utrms = e0x * ham + omtrx
    .....
    if (inc.eq.0) then
      ur = vcx - utrms
      .....
      uvwr2 = ur * ur + vr * vr + wr * wr
      cpu(ir,jr,iter) = 1.0 - uvwr2 / ham2
    else
      call velfdg(iter,icalrec,ir,jr,1,xf,yf,zf,
&      utt,vtt,wtt)
      ur = vcx + utt - utrms
      .....
      uvwr2 = ur * ur + vr * vr + wr * wr
      uvwtrm2 = utrms * utrms + vtrms * vtrms +
&      wtrms * wtrms
      rho = (1.0 + 0.5 * gm1 * (-uvwr2 + uvwtrm2 - 2.0
&      * dphi jr)) ** (1.0 / gm1)
      cpu(ir,jr,iter) = 2.0 / gama / ham2 * (rho ** gama
&      - 1.0)
    endif
  IF (.NOT.inc .EQ. 0) THEN
    uvwr = sqrt(uvwr2)
    ham1 = uvwr / (rho ** (0.5 * gm1))
  ENDIF
ENDDO
ENDDO
RETURN
END

```

Figure 2b. CM Parallel Version - Post-processing.

```

SUBROUTINE GVICAL(IFORS,XRE,YRE,ZRE)
COMMON/VELPG/UPG(20,16,18),VPG(20,16,18),WPG(20,16,18)
.....
IF(IFORS.EQ.0) THEN
  XRE=XA-0.5*DX
  .....
END IF
DO 6 J=1,NY
DO 7 K=1,NZ
DO 8 I=1,NX
XS=XB+(FLOAT(I)-0.5)*DX
.....
XFF=XRE-XS
.....
DIST=SQRT(XFF*XFF+YFF*YFF+ZFF*ZFF)
IF(DIST.GE.FARFD) GOTO 10
IF(ABS(XFF).LT.(0.5*DX).AND.ABS(YFF).LT.(0.5*DY)
+.AND.ABS(ZFF).LT.(0.5*DZ)) THEN
  PDX=0.51*DX
ELSE
  PDX=0.0
END IF
DO 11 IS=1,2
  .....
DO 12 JS=1,2
  .....
DO 13 KS=1,2
  .....
  X2=X*X
  .....
  SQU=SQRT(Y2/X2)
  .....
  ATYZ=ATAN2(Z*SQU,RS)
  .....
  UU(IS,JS,KS)=Y/SQU*ATYZ-0.5*(Y*RSZ+Z*RSY)
  .....
13 CONTINUE
12 CONTINUE
11 CONTINUE
  UT=-(UU(2,2,2)+UU(2,1,1)+UU(1,2,1)+UU(1,1,2)
+  -UU(2,2,1)-UU(2,1,2)-UU(1,2,2)-UU(1,1,1))*OPI4
  .....
  GOTO 14
10 VD3=VOL/(DIST*DIST*DIST)
  UT=OPI4*XFF*VD3
  .....
8 CONTINUE
7 CONTINUE
6 CONTINUE
RETURN
END

```

Figure 3a. Serial Version - Calculating Volume Integrals.

```

SUBROUTINE gvical3( ifors,xre,yre,zre)
COMMON/VELPG/UPG(20,16,18),VPG(20,16,18),WPG(20,16,18)
.....
real ut1(20,16,18),vt1(20,16,18),wt1(20,16,18)
.....
save
if (ifors .EQ. 0) then
  xre = xa - 0.5 * dx
  .....
endif
.....
zff100(:nx,:ny,:nz) = zre-zs100(:nx,:ny,:nz)
dist = sqrt(xff100*xff100+yff100*yff100+zff100*zff100)
where(dist.lt.farfd)
  where (abs(xff100(:nx,:ny,:nz)).lt.(0.5*dx))
    where (abs(yff100(:nx,:ny,:nz)).lt.(0.5*dy))
      where (abs(zff100(:nx,:ny,:nz)).lt.(0.5*dz))
        pdx100(:nx,:ny,:nz) = 0.51*dx
      elsewhere
        pdx100(:nx,:ny,:nz) = 0.0
    elsewhere
      pdx100(:nx,:ny,:nz) = 0.0
  elsewhere
    pdx100(:nx,:ny,:nz) = 0.0
  .....
endwhere
x100(:nx,:ny,:nz) = xre - pdx100(:nx,:ny,:nz)
&-(xs100(:nx,:ny,:nz)-0.5*dx)
.....
squ111(:nx,:ny,:nz)=sqrt((y100(:nx,:ny,:nz)*
& y100(:nx,:ny,:nz))
&/(x100(:nx,:ny,:nz)*x100(:nx,:ny,:nz)))
.....
rs111(:nx,:ny,:nz)=sqrt(x100(:nx,:ny,:nz)*
& x100(:nx,:ny,:nz))
.....
atz111(:nx,:ny,:nz)=atan2(z100(:nx,:ny,:nz)*
& squ111(:nx,:ny,:nz),
.....
&rs111(:nx,:ny,:nz))
.....
uu111(:nx,:ny,:nz)=y100(:nx,:ny,:nz)/
& squ111(:nx,:ny,:nz)
&*atz111(:nx,:ny,:nz)-0.5*(y100(:nx,:ny,:nz)*
.....
where (dist(:nx,:ny,:nz).lt.farfd)
  ut1(:nx,:ny,:nz)=-(uu222(:nx,:ny,:nz)+uu211(:nx,:ny,:nz)
  .....
  elsewhere
    vd3(:nx,:ny,:nz) = vol/(dist(:nx,:ny,:nz)*dist(:nx,:ny,:nz)*
    & dist(:nx,:ny,:nz))
  endwhere
  .....
RETURN
END

```

Figure 3b CM Parallel Version - Calculating Volume Integrals.

```

SUBROUTINE VELFDG(ITER,ICALREC,IR,JR,KR,XFP,YFP,ZFP,
& UTT,VTT,WTT)
COMMON/BLK01/XW(25,13),YW(25,13),ZW(25,13)
.....
IF(ICALREC.EQ.1) THEN
  CALL GVICAL(1,XFP,YFP,ZFP)
  .....
DO 7 JS=1,NY
DO 7 KS=1,NZ
DO 7 IS=1,NX
UTT=UTT+SUPG(IS,JS,KS)*G(IS,JS,KS)*RATIO
.....
7 CONTINUE
END IF
IF(ICALREC.EQ.2) THEN
  .....
DO 6 JS=1,NY
DO 6 KS=1,NZ
DO 6 IS=1,NX
UTT=UTT+SU*G(IS,JS,KS)*RATIO
.....
6 CONTINUE
END IF
IF(ICALREC.EQ.0) THEN
  .....
ID=NX-IR
  .....
DO 8 JS=1,NY
DO 8 KS=1,NZ
DO 8 IS=1,NX
IF(IS.LE.IR) THEN
  I=IS+ID
  SIGI=1.0
ELSE
  IRSD=IS-IR
  I=IR-IRSD+ID
  SIGI=-1.0
END IF
  .....
UTT=UTT+SIGI*UPG(I,J,K)*G(IS,JS,KS)*RATIO
  .....
8 CONTINUE
END IF
RETURN
END

```

Figure 4a. Serial Version - Summing the Volume Integrals.


```

SUBROUTINE velfdg(iter, icalrec, ir, jr, kr, xfp, yfp, zfp, utt,
& vtt, wtt)
COMMON/BLK01/XW(25,13), YW(25,13), ZW(25,13)
.....
real dis31(ny,nz), ds1(ny,nz)
.....
IF (icalrec .EQ. 1) THEN
CALL gvical3(1,xfp,yfp,zfp)
.....
utt = utt + sum(supg(:nx,:ny,:nz) * g(:nx,:ny,:nz) )
.....
END IF
IF (icalrec .EQ. 2) THEN
.....
utt = utt+ sum(su100(:nx,:ny,:nz)*g(:nx,:ny,:nz))
.....
ENDIF
IF (icalrec .EQ. 0) THEN
.....
forall(is=1:ir, js=1:ny, ks=1:nz)
& im2(is, js, ks)= is+idm2(is, js, ks)
forall(is=1:ir, js=1:ny, ks=1:nz)
& sigi(is, js, ks)=1.0
forall(is=ir+1:nx, js=1:ny, ks=1:nz)
& im2(is, js, ks)=ir-is+ir+idm2(is, js, ks)
forall(is=ir+1:nx, js=1:ny, ks=1:nz)
& sigi(is, js, ks)= -1.0
.....
forall(is=1:nx, js=1:ny, ks=1:kr)
& km2(is, js, ks)= ks+kdm2(is, js, ks)
forall(is=1:nx, js=1:ny, ks=1:kr)
& sigk(is, js, ks)= 1.0
forall(is=1:nx, js=1:ny, ks=kr+1:nz)
& km2(is, js, ks) = kr-ks+kr+kdm2(is, js, ks)
forall(is=1:nx, js=1:ny, ks=kr+1:nz)
& sigk(is, js, ks)= -1.0
forall(is=1:nx, js=1:ny, ks=1:nz)
& upg3(is, js, ks)= upg(im2(is, js, ks), jm2(is, js, ks),
& km2(is, js, ks))
forall(is=1:nx, js=1:ny, ks=1:nz)
& vpg3(is, js, ks)=vpg(im2(is, js, ks), jm2(is, js, ks),
& km2(is, js, ks))
forall(is=1:nx, js=1:ny, ks=1:nz)
& wpg3(is, js, ks)= wpg(im2(is, js, ks), jm2(is, js, ks),
& km2(is, js, ks))
utt=utt+sum(sigi(:nx,:ny,:nz)*upg3(:nx,:ny,:nz)*
& g(:nx,:ny,:nz))
.....
ENDIF
RETURN
END

```

Figure 4b. CM Parallel Version-Summing the Volume Integrals.

Table 1. CPU time in seconds for constructing matrices.

size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	0.44	8.25
32-node CM5	0.24	2.34
64-node CM5	0.17	1.20
128-node CM5	0.12	0.65

Table 2. CPU time in seconds for Gaussian elimination.

size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	0.58	33.85
32-node CM5	1.43	8.57
64-node CM5	2.11	7.05
128-node CM5	1.65	7.66

Table 3. CPU time in seconds for post processing.

size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	0.24	4.30
32-node CM5	2.18	9.45
64-node CM5	2.21	9.61
128-node CM5	2.18	9.61

Table 4. Total CPU time in seconds for incompressible flow.

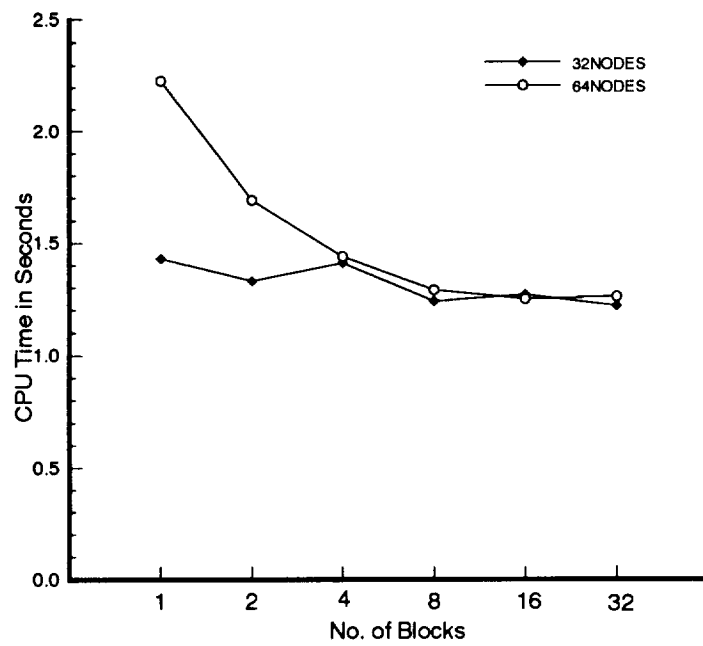
size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	1.33	46.60
32-node CM5	3.86	20.55
64-node CM5	4.53	18.01
128-node CM5	4.00	18.08

Table 5. CPU time in seconds for evaluating volume integrals.

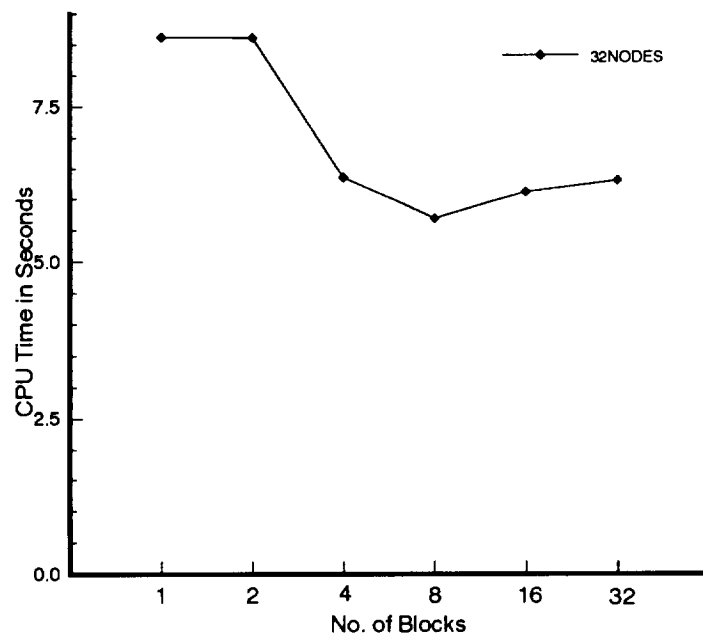
size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	1.48	5.60
32-node CM5	0.11	-
64-node CM5	0.07	0.07
128-node CM5	0.04	0.04

Table 6. Total CPU time in seconds for 6 iterations.

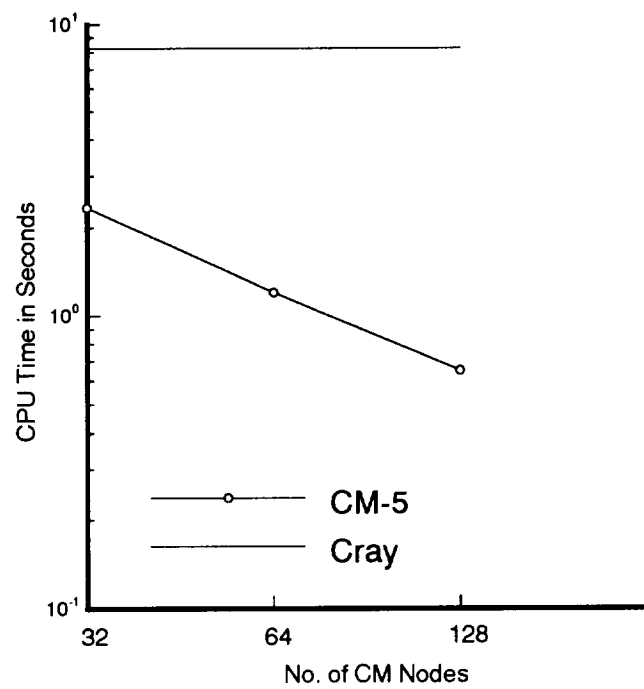
size, N=	288(=24x12)	1152(=48x24)
Cray-YMP	86.0	412.0
32-node CM5	1416.0	-
64-node CM5	1300.0	1634.0
128-node CM5	1202.0	1396.0



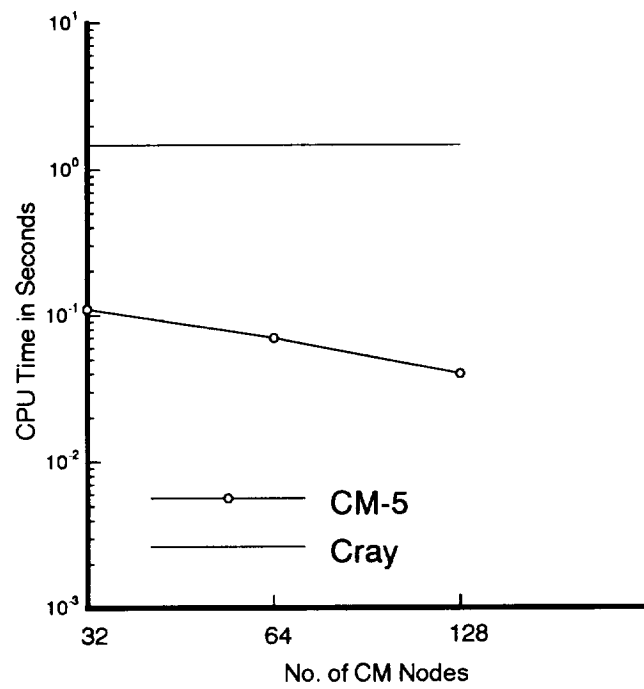
Graph 1. Effect of number of blocks in Gaussian elimination, $N=24 \times 12$.



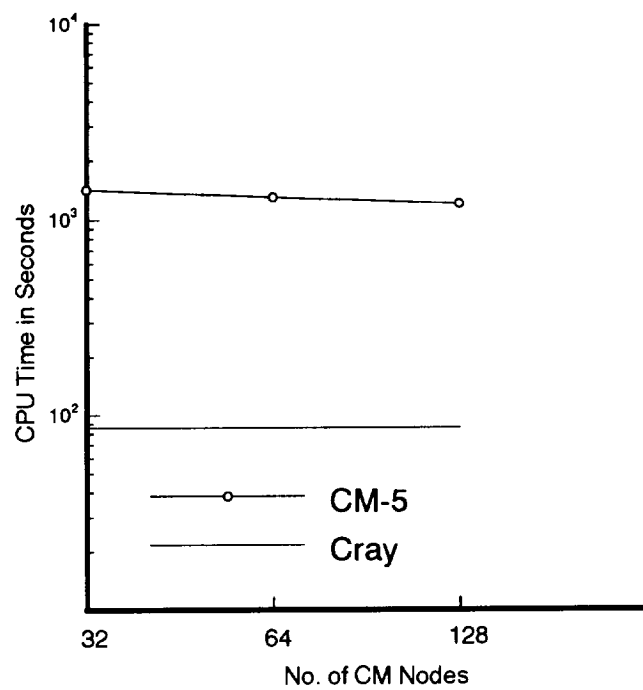
Graph 2. Effect of number of blocks in Gaussian elimination, $N=48 \times 24$.



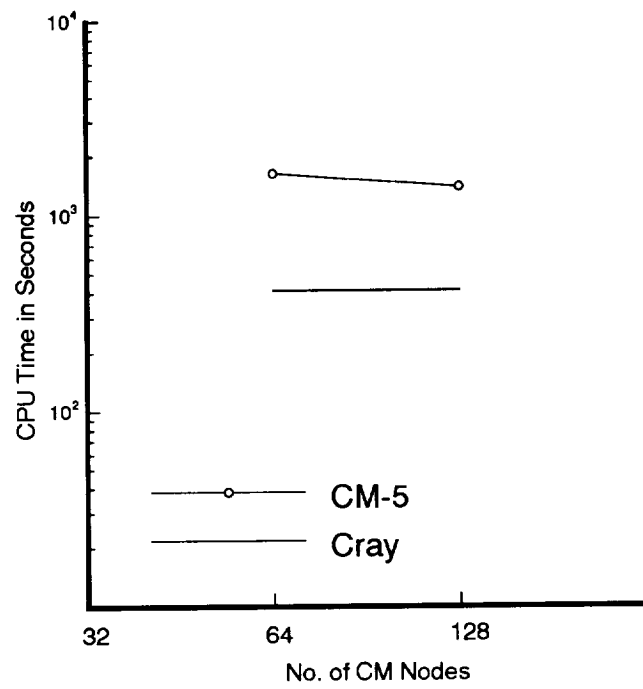
Graph 3. CPU time for constructing matrices, $N=48 \times 24$.



Graph 4. CPU time for evaluating volume integrals, $N=24 \times 12$.



Graph 5. CPU time for compressible flows with 6 iterations, $N=24 \times 12$.



Graph 6. CPU time for compressible flows with 6 iterations, $N=48 \times 24$.

REFERENCES

1. Anderson, John D., Fundamental of Aerodynamics, 2nd ed., McGraw-Hill, USA, 1991.
2. Chriske, D.M. and Boguez, E.A., Performance Evaluation of the Connection Machine for Panel Method Calculations, AIAA Paper 91-0439, 1991.
3. Hillis, Daniel W., The Connection Machine, Scientific American, Vol. 256, June 1987, pp. 108-115.
4. Hu, H. and Jackson, Isaac T., Comparative study of Computational Performance of CM-2 and Cray-YMP for Boundary Element Computations, *Boundary Element Abstracts Journal and Newsletter*, 3(5), 1992 pp. 185-191.
5. Hu, H., Study of Integral Equation Methods for Transonic Flow Calculations, *Engineering Analysis with Boundary Elements*, 11(2), 1993, pp.101-107.
6. Kandil, O. A. and Yates, E. C., Jr., Computation of Transonic Vortex Flow Past Delta Wings - Integral Equation Approach, AIAA Paper 87-0421, 1987.
7. Logan, T. G. and Hu, H., Performance Study of 3D Integral Equation Computations on Massively Parallel Computer, in *Boundary Element Methods in Fluid Dynamics II*, (Eds.: H. Power, C. A. Brebbia and D. B. Ingham), Computational Mechanics Publication, Southampton, Boston, 1994, pp. 85-91.
8. CM-5, Technical Summary, Thinking Machines Corporation, Cambridge, MA, 1992.
9. CM FORTRAN Reference Manual, Preliminary Documentation for Version 2.0 Beta, Thinking Machines Corporation, Cambridge, MA, 1992.
10. CMSSL for CM FORTRAN: CM-5, Vol. 1, Version 3.1 Thinking Machines Corporation, Cambridge, MA, 1993.

ABSTRACT

**PERFORMANCE ANALYSIS OF THREE DIMENSIONAL
INTEGRAL EQUATION COMPUTATIONS ON
A MASSIVELY PARALLEL COMPUTER**

**Student: Terry G. Logan, Department: Mathematics
Degree: Master of Science, August 1994**

The purpose of this study is to investigate the performance of the integral equation computations using numerical source field-panel method in a massively parallel processing (MPP) environment. A comparative study of computational performance of the MPP CM-5 computer and conventional Cray-YMP supercomputer for a three-dimensional flow problem is made. A serial FORTRAN code is converted into a parallel CM-FORTRAN code. Some performance results are obtained on CM-5 with 32, 64, 128 nodes along with those on Cray-YMP with a single processor. The comparison of the performance indicates that the parallel CM-FORTRAN code near or out-performs the equivalent serial FORTRAN code for some cases.